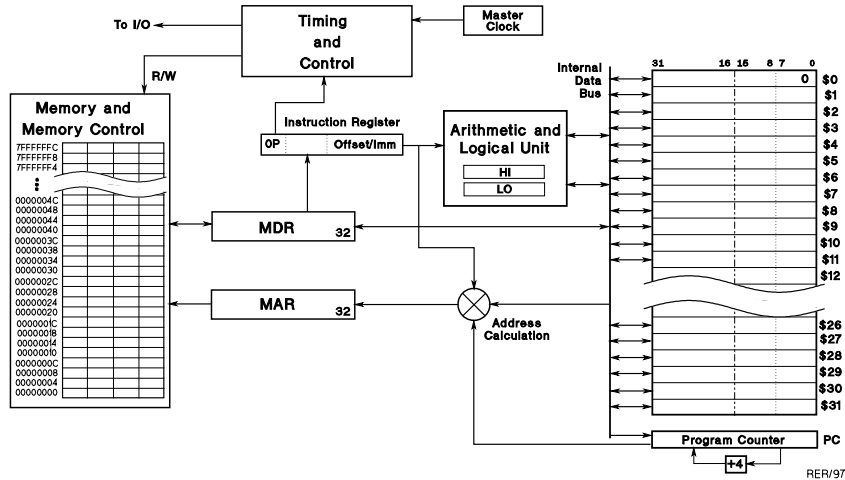
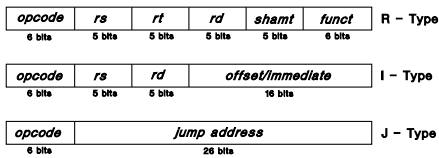


# MIPS R2000 – Instruction Reference Card

## Programmer's Model of the MIPS R2000 Microprocessor



### • MIPS Binary Instruction Formats



Fields in the instruction formats are:

- opcode** - (6 bits)
- rs** - (5 bits) first source register
- rt** - (5 bits) second source register
- rd** - (5 bits) destination register
- shamt** - (5 bits) shift amount
- funct** - (6 bits) additional opcode
- off/imm** - (16 bits) immediate value/address offset
- jmp addr** - (26 bits) jump address

### • Register Conventional Usage

Name	Register	Description/Usage
	\$0	Hardwired constant 0
\$at	\$1	Reserved for assembler
\$v0-\$v1	\$2-\$3	Return results from functions
\$a0-\$a3	\$4-\$7	Used for subroutine arguments.
\$t0-\$t7	\$8-\$15	"Temporary," not saved across a call
\$s0-\$s7	\$16-\$23	"Saved," saved across a call
\$t8-\$t9	\$24-\$25	More temporary
\$k0-\$k1	\$26-\$27	"Kernel" - reserved for OS use
\$gp	\$28	Pointer to global data area
\$sp	\$29	Stack Pointer
\$fp	\$30	Frame pointer
\$ra	\$31	Return address of subroutine

### • Addressing Modes Used with Load/Stores

Mode	Syntax	Effective Address
Relative	RSYMB	EA = [PC] + offset
Register Indirect	(\$n)	EA = [\$n]
Base	offset(\$n)	EA = [\$n] + offset

### • Load/Store, Data Movement Instructions

Description	Opcode	Operand	Format
Load Address	la	rd, mem	†
Load Byte	lb	rd, mem	I
Load Byte Unsigned	lbu	rd, mem	I
Load Halfword	lh	rd, mem	I
Load Half Unsigned	lhu	rd, mem	I
Load Word	lw	rd, mem	I
Load Immediate	li	rd, imm	†
Load Upper Immed	lui	rd, imm	I
Store Word	sw	rs, mem	I
Store Halfword	sh	rs, mem	I
Store Byte	sb	rs, mem	I
Move	move	rd, rs	†

### • Arithmetic/Logical Instructions

Description	Opcode	Operand	Format
Add	add	rd, rs, rt	R
Add Immediate	addi	rd, rs, imm	I
Add Unsigned	addu	rd, rs, rt	R
Add Unsigned Immed	addiu	rd, rs, imm	I
Divide	div	rd, rs, rt	†
Divide Unsigned	divu	rd, rs, rt	†
Multiply	mul	rd, rs, rt	†
Multiply Unsigned	mulu	rd, rs, rt	†
Negate	neg	rd, rs	†
Remainder	rem	rd, rs, rt	†
Subtract	sub	rd, rs, rt	R
Subtract Unsigned	subu	rd, rs, rt	R
And	and	rd, rs, rt	R
And Immediate	andi	rd, rs, imm	I
Not	nor	rd, rs, rt	R
Not	not	rd, rs	†
Or (Inclusive)	or	rd, rs, rt	R
Or Immediate	ori	rd, rs, imm	I
eXclusive OR	xor	rd, rs, rt	R
eXclusive OR Immed	xori	rd, rs, imm	I

## • Branch/Jump Instructions

Description	Opcode	Operand*	Fmt
Branch (always)	<b>b</b>	label	†
Jump	<b>j</b>	label	J
Branch if Equal	<b>beq</b>	rs, rt, label	I
Branch if Not Equal	<b>bne</b>	rs, rt, label	I
Branch if Greater	<b>bgt</b>	rs, rt, label	†
Branch if Greater or Equal	<b>bge</b>	rs, rt, label	†
Branch if Greater Unsigned	<b>bgtu</b>	rs, rt, label	†
Branch if Greater or Equal Unsigned	<b>bgeu</b>	rs, rt, label	†
Branch if Less Than	<b>blt</b>	rs, rt, label	†
Branch if Less or Equal	<b>ble</b>	rs, rt, label	†
Branch if Less Unsigned	<b>bltu</b>	rs, rt, label	†
Branch if Less or Equal Unsigned	<b>bleu</b>	rs, rt, label	†
Branch if Equal to Zero	<b>beqz</b>	rs, label	†
Branch if Not Equal to Zero	<b>bnez</b>	rs, label	†
Branch if Greater Than Zero	<b>bgtz</b>	rs, label	I
Branch if Greater or Equal to Zero	<b>bgez</b>	rs, label	I
Branch if Less Than Zero	<b>bltz</b>	rs, label	I
Branch if Less or Equal to Zero	<b>blez</b>	rs, label	I

\* In the three operand branch instructions, *rt* can be replaced with a constant, i.e., operands can be *rs,imm,label*. All such instructions are pseudoinstructions.

† - pseudoinstruction

## • Shift and Rotate Instructions

Description	Opcode	Operand	Fmt
Rotate Left	<b>rol</b>	rd, rs, rt	†
Rotate Right	<b>ror</b>	rd, rs, rt	†
Shift Left Logical	<b>sll</b>	rd, rs, sa	R
Shift Left Log Variable	<b>sllv</b>	rd, rs, rt	R
Shift Right Logical	<b>srl</b>	rd, rs, sa	R
Shift Right Log Variable	<b>srlv</b>	rd, rs, rt	R
Shift Right Arithmetic	<b>sra</b>	rd, rs, sa	R
Shift Right Arith Variable	<b>srav</b>	rd, rs, rt	R

## • Comparison Instructions

Description	Opcode	Operand	Fmt
Set if Equal	<b>seq</b>	rd, rs, rt	†
Set if Not Equal	<b>sne</b>	rd, rs, rt	†
Set if Less Than	<b>slt</b>	rd, rs, rt	R
Set if Less or Equal	<b>sle</b>	rd, rs, rt	†
Set if Greater Than	<b>sgt</b>	rd, rs, rt	†
Set if Greater or Equal	<b>sge</b>	rd, rs, rt	†
Set if Less Unsigned	<b>sltu</b>	rd, rs, rt	R
Set if Less or Equal Unsigned	<b>sleu</b>	rd, rs, rt	†
Set if Greater Unsigned	<b>sgtu</b>	rd, rs, rt	†
Set if Greater or Equal Unsigned	<b>sgeu</b>	rd, rs, rt	†
Set if Less Than Immed	<b>slti</b>	rd,rs,imm	I
Set if Less Than Imm Unsigned	<b>sltiu</b>	rd,rs,imm	I

## • Subroutine Instructions

Description	Opcode	Operand	Format
Jump and Link (Call)	<b>jal</b>	label	J
Jump Register	<b>jr</b>	rs	R

## • Miscellaneous Instructions

Description	Opcode	Operand	Format
No Operation	<b>nop</b>		†
System Call	<b>syscall</b>	code	R

## • Assembler Syntax

```
label: mnemonic operands # comment
```

## • Assembler Directives - SPIM Subset

Directive	Description
<b>.text</b>	Put next code into <i>text</i> section
<b>.data</b>	Put next code into <i>data</i> section
<b>.globl name</b>	Make <i>name</i> be a global symbol
<b>.space n</b>	Allocate <i>n bytes</i> of space
<b>.word val1,val2, ...</b>	Allocate one word for each value
<b>.half val1,val2, ...</b>	Allocate a halfword for each value
<b>.byte val1,val2, ...</b>	Allocate a byte for each value
<b>.ascii "string"</b>	Allocate space for the ASCII chars
<b>.asciiz "string"</b>	Allocate space for string, NULL terminated
<b>.align n</b>	Start next on 2 <sup>n</sup> byte boundary

## • SPIM System Services

Routine	Code in \$2	Arguments	Result
<b>print_int</b>	1	\$4 = integer	
<b>print_str</b>	4	\$4 = addr of string	
<b>read_int</b>	5		integer in \$2
<b>read_str</b>	8	\$4 = addr of buffer (memory) \$5 = max length of str + 1	str in buffer
<b>malloc</b>	9	\$4 = # of bytes desired	address in \$2
<b>exit</b>	10		